FPGA Implementation of An Array Processor

M.Sc. Firas M. Naif

Received on:27/6/2005
Accepted on:26/1/2006

Abstract

This paper describes how to map the array processor to the Xilinx Field Programmable Gate Array (FPGA) Chip. The design is connected to personal computer and tested as a matrix multiplication using a high level (Visual Basic) program.

The developed convolution architecture was captured through the use of the Very High Scale Hardware Description Language (VHDL). Xilinx Foundation series (4.1i) Computer Aided Design (CAD) software package was used to synthesize and implement the architecture of the array processor to the *FPGA* chip. Before the architecture was prototyped into the prototyping board for experimental testing, the architecture was functionally and performance validated through Hardware Description Language (HDL) software simulation via post-synthesis and post-implementation software simulations.

الخلاصة

ان هذا البحث يوصف كيفية تمثيل معالج متعدد الخلايا في رقاقة المصفوفات البرمجية (FPGA) لشركة (Xilinx) للالكترونيات الدقيقة.
قد تم ربط التصميم الى الحاسبة الشخصية من اجل اختباره كضارب للمصفوفات (٢x٢) باستخدام لغة البرمجة ذات المستوى العالي (البيسك المرئي).
تم تصميم و تطوير معمارية المعالج باستخدام لغة وصف التصميم العالية السرعة (VHDL) كبيئة برمجية في حين تمت عملية التركيب و التنفيذ بأستخدام برنامج (Foundation Series 4.1i).
قبل اختبار التصميم عمليا تمت عملية محاكات برمجية (Simulations Software) للتصميم من اجل اختبار كفائة التصميم قبل تنفيذه.

Keywords

Array processor, Field Programmable Gate arrays (FPGA), Matrix Multiplication, VHDL

1. Introduction

An array processor is a processor that performs computations on large arrays of data. The term is used to refer to two different types of processors. An attached array processor is designed as a peripheral for a conventional host computer, and its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications. It achieves high performance by means of parallel processing with multiple functional units. It can be programmed by the user to accommodate a variety of complex

Control and System Department, University of Technology, Baghdad-Iraq.

arithmetic problems such as matrix multiplication, inversion, and L-U decomposition.

A Single Instruction Multi Data (SIMD) array processor is a processor that has a single –instruction multiple-data organization. It is a processor with multiple processing units operating in parallel. The processing units are synchronized to perform the same operation under the control of a common control unit.

Although both types of array processors manipulate vectors, their internal organization is different [1].

2. Introduction To Vhdl

The popularity of using hardware description language (HDL) for designing digital circuits began in the middle of 1990s when commercial synthesis tools become available. Two popular HDLs used by many engineers today are VHDL and Verilog. VHDL, which stand for VHSIC Hardware Description Language (VHSIC, in turn, stands for Very High Speed Integrated Circuit) [2].

The VHDL provides the digital designer with a mine of describing a digital system at a wide range of levels of abstraction [3].

VHDL, in many respects, is similar to a regular computer programming language, such as C. for example, it has constructs for variable assignments, conditional statements, loops, and functions. In a computer program language, a compiler is used to translate the high-level source code to machine code. In VHDL, however, a synthesizer is used to translate the source code to a description of the actual hardware circuit that implements the code. Accurate functional and timing simulation of the code is also possible in order to test the correctness of the circuit.

VHDL used by over 10,000 designers at such hardware venders as Sun, Microsystem, Apple Computer, and Motorola [4][5].

3. Processor Design

In this section it is required to design a matrix multiplication array processor .In general, this processor is a two-dimensional pipeline with multiple data-flow streams for high-level arithmetic computations. It is pipelined in three data-flow directions for the repeated multiplication of pairs of compatible matrices. The basic building blocks in the array are the M cells. Each M cell performs an additive inner-product operation as illustrated in figure (1).

Figure (2) shows the schematic diagram, which is a diagram showing how to logic gates are connected together inside each M cell [6].

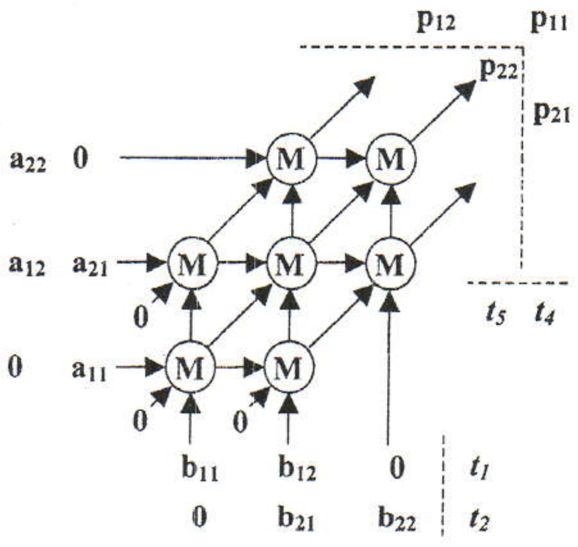


Figure (1) The matrix multiplication array processor

From figure (2), it is clear how many signal lines each M cell has. So it has three input operands a, b, and c and three outputs x, y and z. The operation of each cell can be described simply by saying that:

$x = a \quad , \quad y = b$

$z = (a.b) + c$

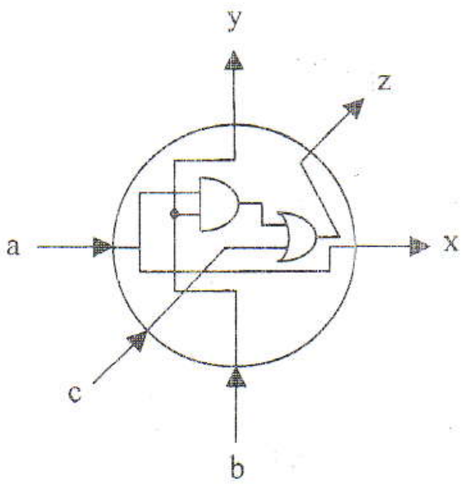


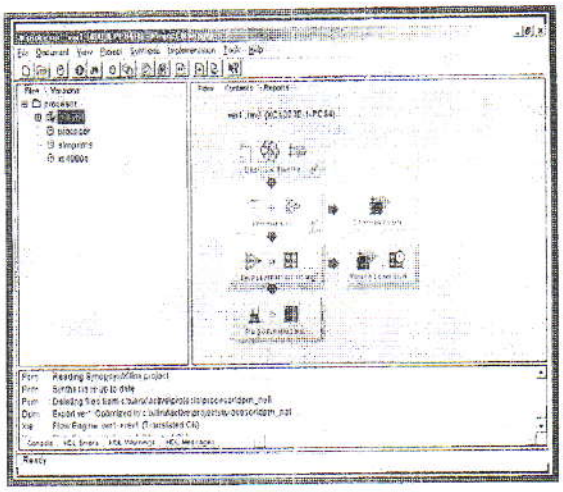
Figure (2) The Logic gates Schematic diagram of the M cell

Latches (registers) are used at all input-output terminals and all interconnecting paths in the array pipeline. All latches are synchronously controlled by the same clock (ck). Adjacency between cells is defined in the three orientations: horizontal, vertical and diagonal (45°) directions.

4. Xilinx Foundation Series 4.1I WITH VHDL PROGRAM

One of the most interesting aspects of the FPGA technology is that its implementation (hardware) side is totally determined by its description (software). [7]

Figure (3) shows Xilinx Foundation series (4.1i) CAD software package that is used to synthesize and implement the architecture of the array processor to the FPGA chip.


File Document View Project Synthesis Implementation Tools Help
Files Versions
processor
Flow Contents Reports
Ready

```
Pcm  Reading Synopsys/Xilinx project
Pcm  Synthesis is up to date
Pcm  Deleting files from [illegible]
Dpm  Export ver: Optimized in [illegible]
Xip  Flow Engine [illegible] (Translated CA)
```

Figure (3) The Xilinx Foundation series (4.1i) CAD software package

The first step that has been followed in implementing the processor design is *Design Entry*.

The design entry in the proposed design has been done using VHDL programming language. So, VHDL is used to create an entity that performs each cell operation. When describing a circuit at this level, you would write basically the same thing as in the arithmetic description, except that you have to use the correct syntax required by the VHDL.

The description of the array processor architecture using VHDL is shown bellow:

```
------ Cell ------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY cell IS PORT(
    a, b, c, k: IN STD_LOGIC;
    x, y, z: OUT STD_LOGIC);
END cell;
ARCHITECTURE Dataflow OF
cell IS
```

```
BEGIN
  If  k='1' then
   x <= a;
   y <= b;
   z <= (a AND b) OR c;
end if;

END Dataflow;
------ Array processor ----
--
LIBRARY IEEE;
USE
IEEE.STD_LOGIC_1164.ALL;
ENTITY Array processor IS
PORT(
   x1, x2, x3: IN
STD_LOGIC;
   y1, y2, y3: IN
STD_LOGIC;
   c1, c2, c3, ck: IN
STD_LOGIC;
   P1, p2, p3: OUT
STD_LOGIC);
END Array processor;
ARCHITECTURE Structural OF
Array processor IS
   COMPONENT cell PORT(
   a, b, c, k: IN
STD_LOGIC;
   x, y, z: OUT STD_LOGIC);
END COMPONENT;

SIGNAL xx1, xx2, xx3, xxx1,
       xxx2, xxx3, xxxx2,
       yy1, yy2, yy3, yyy1,
       yyy2, yyy3, yyyy2,
       p4, p5, p6, p7:
STD_LOGIC;

BEGIN

U1: cell PORT MAP(x1, y1,
c1, ck, xx1, yy1, p7);
U2: cell PORT MAP(x2, yy1,
c3, ck, xx2, yyy1, p6);
U3: cell PORT MAP(xx1, y2,
c2, ck, xxx1, yy2, p5);
U4: cell PORT MAP(xx2, yy2,
p7, ck, xxx2, yyy2, p4);
U5: cell PORT MAP(x3, yyy2,
p6, ck, xx3, yyyy2, p3);
U6: cell PORT MAP(xxx2, y3,
p5, ck, xxxx2, yy3, p2);
U7: cell PORT MAP(xx3, yy3,
p4, ck, xxx3, yyy3, p1);

END Structural;
----------------------------
```

The second step is *Design Synthesis*. In this step, the question now is how does the program that describes the operation of each cell actually get converted to the physical circuit. The synthesizer is used to translate the VHDL program into the circuit netlist.

The design has been synthesized; a physical netlist of the design is produced consisting of primitive structures that are technology-specific. For this effort, design were targeted to Xilinx (XC4000E) FPGA type (PC84CKM9721). The resulting netlist is then used to produce the configuration bit steam that programs the FPGA device.

The next step in the design flow is *Simulation*. This activity can verify the functional properties of a synthesized design. This simulation is an ideal time to catch design faults such as incorrect module annotation problems in the data flow and incomplete design descriptions.

The designed circuit is simulated by simulating the inputs (x1, x2, x3, y1, y2, y3, c1, c2, c3, and ck) with a known sequence of values and examining the outputs (p1, p2, and p3).

Figure (4) illustrate the simulation of multiplying two matrixes A & B with the immediate values that shown bellow:

$$P = A \times B$$

$$\begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

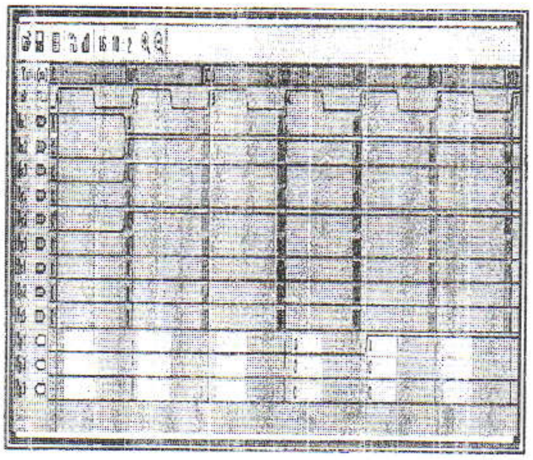
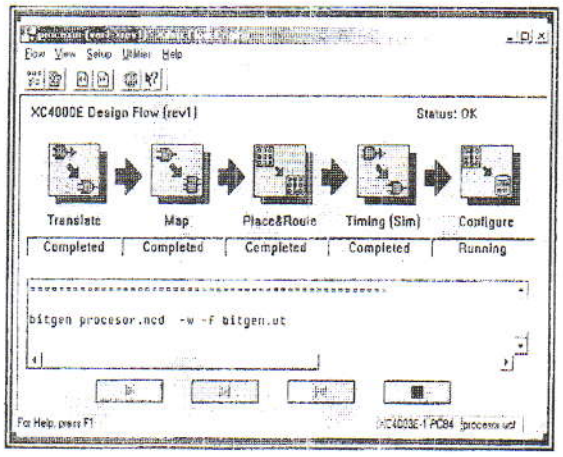


Figure (4) The simulation of multiplying two matrixes A & B

After feeding the two matrices into the array in the horizontal and vertical directions. Two clock periods are needed for inputting the matrix entries: one row at a time for the A matrix and one column at a time for B matrix. Dummy zero inputs are marked at unused input lines. After two clock periods, three results (p_{12}, p_{11}, p_{21}) will be produced. Finally the last product result (p_{22}) will be produced after the fifth clock period. In general, to multiply two $(n \times n)$ matrices requires $(3n^2 - 4n + 3)$ M cells. It takes $(3n - 1)$ clock periods to complete the multiply process. [6] So in our design the number of M cells is ***seven*** cells and ***five*** clock periods is required to complete the multiplication.

The forth step is *Design Implementation*. It concerned with the exact section of the circuit primitives and their placement and routing with some given constraints. This step has several operations like completes the hardware design, translates the gate level design into hardware primitives available in (XC4000E) FPGA, assigns the design to physical locations on the chip and route the connections between them, timing information about the design, and determines the configuration bits to implement the design. Figure (5) illustrates the XC4000E design flow implementation.

After the completion of implementation step the Programming step is began to generate a programming file, which specifies the operation of the FPGA device. The data of this file is stored in an E^2PROM. When the power is applied to the FPGA chip, the storage cells are loaded automatically from the memory.



XC4000E Design Flow (rev1) Status: OK

Translate	Map	Place&Route	Timing (Sim)	Configure
Completed	Completed	Completed	Completed	Running

```
bitgen procesor.ncd  -w -f bitgen.ut
```

For Help, press F1 XC4003E-1 PC84 procesor.ud

Figure (5) The XC4000E design flow implementation

At the end, the Floor Planner can be used to know more information about the design connectivity and resource requirements, target FPGA resource layout, and the design mapping via location constraints, as shown in figure (6).

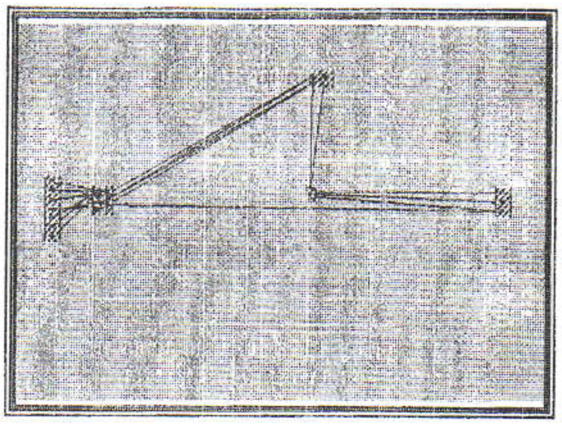
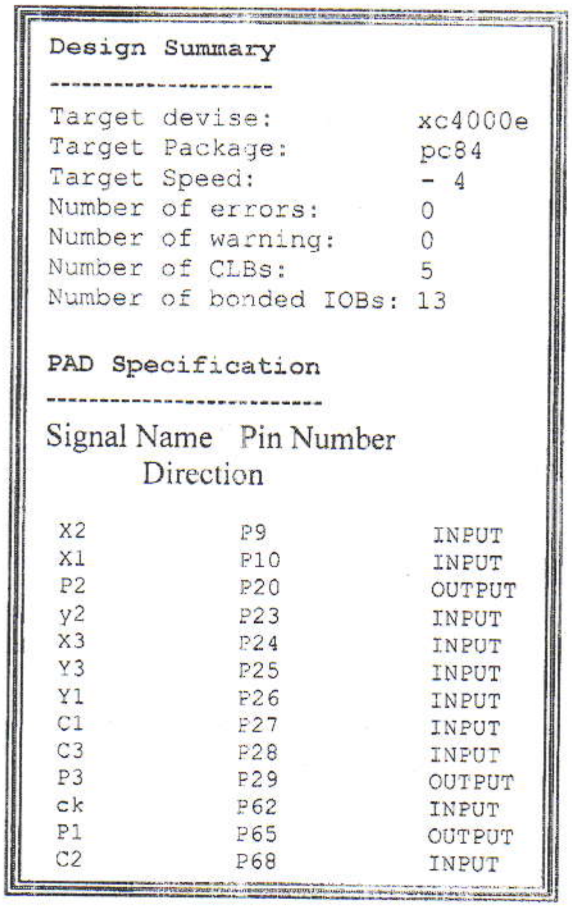


Figure (7) shows the XC4000E design summary and the pad specification that are generated for showing all the signal name and the equivalent pin number.



```
Design Summary
--------------------
Target devise:          xc4000e
Target Package:         pc84
Target Speed:           - 4
Number of errors:       0
Number of warning:      0
Number of CLBs:         5
Number of bonded IOBs:  13

PAD Specification
-------------------------
Signal Name   Pin Number
      Direction

X2        P9      INPUT
X1        P10     INPUT
P2        P20     OUTPUT
y2        P23     INPUT
X3        P24     INPUT
Y3        P25     INPUT
Y1        P26     INPUT
C1        P27     INPUT
C3        P28     INPUT
P3        P29     OUTPUT
ck        P62     INPUT
P1        P65     OUTPUT
C2        P68     INPUT
```

Figure (7) The design summary and pad report.

5. Array Processor Hardware Design

Figure (8) shows the schematic sheet of the overall hardware design and how to connect the FPGA chip to the E^2PROM.

The XC4000E support system clock rate up to 70 MHz and internal performance in excess of 150 MHz.

In our design, the master parallel up mode is used by selecting <100> on the mode pins (M2, M1, M0) of the FPGA. So that the FPGA will directly address in industry-standard byte-wide E^2PROM (X28HC64P), and accept eight data bits just before incrementing the address output [8]. Figure (8) also shows the inputs and outputs lines, which are connected to the standard parallel port connector in order to connect the designed hardware to the personal computer (PC) in the testing phase.

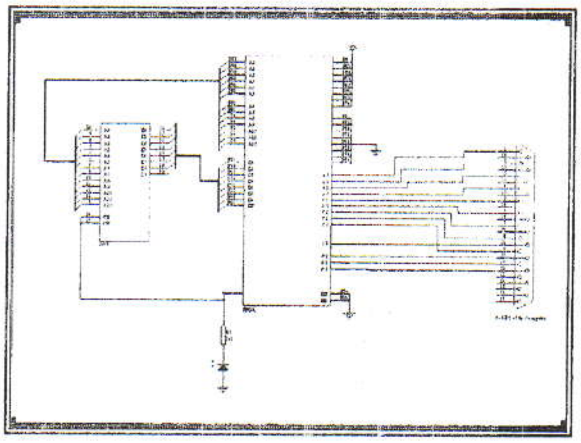


Figure (8) The schematic sheet of the overall hardware

Figure (9) illustrates the photo picture of the hardware design that have been implemented and tested practically.

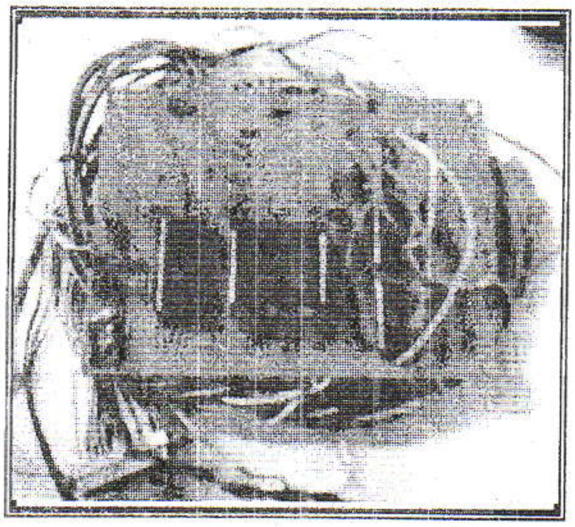


Figure (9) The Hardware Photo Picture.

6. Testing Phase of The Designed Array Processor

When the hardware implementation of the array processor is completed, the testing phase begins. A simple program has been written using a high level language **(Visual Basic 6)** in order to test the hardware design. This program provides the ability to the user to interact with the hardware design, so the user can apply the values of each matrix A and B to the hardware and then read the result from it.

At the same time the program will calculate the result theoretically and compare the two results. If there is equal, the program will show an **Ok** massage otherwise **Error** massage will appear. Figure (10) shows the array processor-testing program.

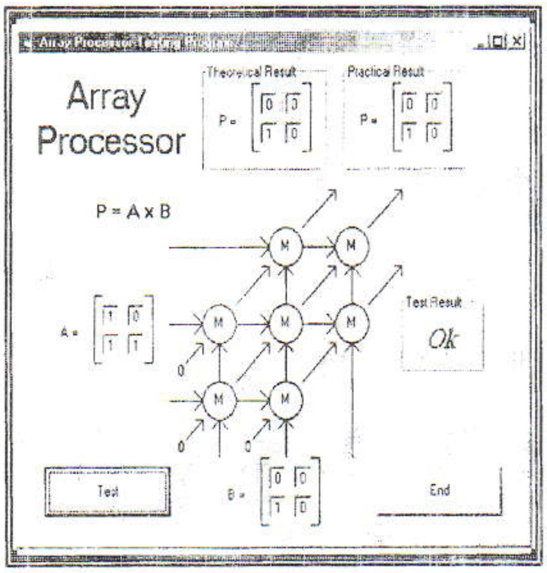


Figure (10) The array processor-testing program.

This operation will be done after inserting of the matrices values in the suitable place and clicking on the *Test* command.

The *End* command is used to exit from the program.

7. Conclusions

In this paper, a complete matrix multiplication array processor has been designed and implemented using FPGA technology.

For the matrix (2×2), it is clear from the simulation phase that the array processor needed 400 nsec to complete the multiplication operation. This time will be added to the signal propagation time from the PC to the FPGA and from the FPGA to the PC.

FPGAs provide a low cost of implementation with high flexibility for tolerating permanent faults in the hardware circuit.

Using FPGA technology in our design has also enabled us to replace a number of logic chips in the existing hardware circuit, and reduce the area of circuit board that carry such chips.

For expansion, it is easily to modify the program of the array processor to multiply $(n \times n)$ matrixes.

The FPGAs have provided a fast development cycle, and the flexibility to alter the design for bug fixes and functional enhancements.

8. References

[1] Morris M. Mano, Computer System Architecture, Prentice Hall PTR, third edition, 1993.

[2] Eric Crabill, Introduction to Verilog for Use with FPGAs, 2004.

[3] Karen Pamell and Nick Menta, Programmable Logic Design Hand Book, Wiley-Interscience, 2002.

[4] J. Bhasker, A VHDL Primer, Prentice Hall PTR, fourth edition, 2002.

[5]John F. Wakerly, Field-Programmable Gate Array, 1999.

[6] Kai Huang, Advance Computer Architecture: Parallelism, Scalability, and Programmability, McGraw-Hill Book Company, 1999.

[7] Xilinx Co., Foundation series (4.1i) Software Manual,
At:
http://toolbox.xilinx.com/docsan/xilinx4/pdf/manuals.pdf

[8] Xilinx Co., XC4003e and XC400X Series Field Programmable Gate Array data Book, 2000
At:
http://www.cse.wustl.edu/~jwhite/class/coe455/docs/4ke-electrical.pdf